Computer Lab 08

In this lab you will write a class implementing the composition method for triangular densities. An instance of the class you write can be retrieved from the `RandomFactory` class you have been using in your other labs for built-in random variates (such as Exponential).

Concepts

- Composition method
- Implementing an instance of `RandomVariate`
- Histograms in Simkit

Description

This lab consists of the following parts: (1) Write the `TriangularVariate` class; (2) Test the `TriangularVariate` class; and (3) Write the code to create histograms

1. Write TriangularVariate Class

The `TriangularVariate` class will use the composition method to generate a triang(a, b, c) random variate. Recall that the algorithm is

Generate U, V ~ Un(0, 1)
if (V < (c - a) / (b - a))
Return a + (c - a) $\sqrt{U}$
else
Return b - (b - c) $\sqrt{1-U}$

This algorithm should be implemented in the `generate()` method. For a class to be "found" by the RandomFactory, it must implement the `simkit.data.RandomVariate` interface. The `RandomVariate` interface has the following methods:

```
void setSeed(long);
void resetSeed();
void setParameters(Object[]);
Object[] getParameters();
double generate();
```

Constructor

The constructor should have signature `(Object[], long)`. This is the most general type of constructor that can be found by `RandomFactory` and should suit most purposes. Declare an instance variable of type `RandomNumber` called `rng` that will be used for your uniform (0,1) variates.

You will need three instance variables to hold the left, right, and center values; call these `left`, `right`, and `center`, respectively.[1] The setter need to be handled a bit differently than usual; because `RandomFactory` passes the parameter values via `setParameters(Object[])`, which you must write. For subclassing purposes, you should declare these four instance variables to be `protected` rather than `private`.

1. Yes, very imaginative!

setSeed(long), resetSeed() Methods

These methods should simply pass the argument to `setSeed(long)` of the `RandomNumber` instance variable.

setParameters(Object[])

This method needs to unwrap the three values in the `Object[]` argument and store them in the three instance variables (`left`, `right`, `center`). To be safe, you should only do this if (1) The argument is not `null`; (2) It has `length` equal 3; and (3) All of its elements are instances of `Number`.[2] Check each `Object`'s type with "`instanceof Number`".[3] Throw an `IllegalArgumentException` from `setParameters()` if all these conditions are not met.

getParameters()

You should also write `public Object[] getParameters()` to return an `Object[]` consisting of the three values wrapped in `Double` objects.

generate()

The generate() method is the main purpose for writing the class in the first place. Implement the algorithm

Other Constructors

After you have tested your class, it ise useful to write at least one other constructor that has signature `(Object[], RandomNumber)`. This should set the parameters, as with the first constructor, but also set the `RandomNumber` instance variable to the second argument. Another useful constructor simply has signature (`Object[]`) that gets its RandomNumber from RandomFactory with the default seed (i.e. with `RandomFactory.getRandomNumber()`).

Test the TriangularVariate Class

After you get the `TriangularVariate` class to compile, write TestGenerate as a pure execution class to test it out. Use `RandomFactory` to retrieve an instance with parameters (1.0, 2.5, 1.5) as follows:

```
Object[] parameters = new Object[] {new Double(1.0), new Double(2.5), new Double(1.5)};
long seed = RandomStream.STREAM[4];
RandomVariate triangle =
        RandomFactory.getRandomVariate("oa3302.TriangularVariate", parameters, seed);
```

Note that unlike previous usage, you must give the fully qualified name of your `RandomVariate` class to `RandomFactory`. The first five generated variates should be:

```
2.186939575801437
2.3945392174305744
1.6134242547059288
1.4371963823097882
1.4645334820214746
```

2. Number is the superclass of Integer, Double, Float, etc.
3. For example, (`param[0] instanceof Number`) evaluates to true if the object param[0] is an Integer, Double, Float, etc and false otherwise.

Verify that you get the same numbers when you get the `RandomVariate` instance from RandomFactory with a third parameter in `getRandomVariate()` equal to `RandomStream.STREAM[0]`.

Write the Code to Create Histograms

To see the results of your class visually, use the following code:[4]

```
Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
Rectangle window = new Rectangle(400, 300);
       CloseableDataWindow cdw = new CloseableDataWindow("Triangular Variate Using Composition");
cdw.setBounds((screen.width - window.width) / 2, (screen.height - window.height) / 2,
             window.width, window.height );
GraphStat gs = new GraphStat("Triangular", 0.0);
cdw.add(gs.initHistogram(true, 1.0, 2.5, 100));
cdw.setVisible(true);
```

In the above code fragment, the first two lines establish the dimensions of the screen and of the data window. The `CloseableDataWindow` class is the shell for displaying the histogram and the setBounds() command places the window at the center of the screen. The `GraphStat` instance produces the histogram, itself with the `initHistogram()` method. The two arguments to GraphStat's constructor are not used in this lab, but are necessary to instantiate a `GraphStat` (the `String` and `double` can be arbitrary, in fact).

The arguments to GraphStat's initHistogram() method are as follows:

- `boolean` - `true` if histogram is animated, `false` if not
- `double` - lower limit of histogram
- `double` - upper limit of histogram
- `int` - number of cells in histogram

Finally, to generate the output, write the following loop:

```
for (int i = 0; i < numberToGenerate; i++) {
       gs.sample(0.0, triangle.generate());
       cdw.repaint();
}
```

The `sample()` method of `GraphStat` requires a double as its first argument for reasons that do not apply to today's lab. The second argument is the new observation; `GraphStat` will put it in the appropriate bin and update the count. The `repaint()` method will redraw the histogram after the new observation.

Acceptance/Rejection Method

Now write a class called `TriangularARVariate` that generates triang(a,b,c) random variates using the acceptance/rejection method. You can subclass `TriangularVariate` and just override the generate() method.[5] Since you declared your instance variables in `TriangularVariate` to be `protected`, you can use them in this class.

---

4. You will have to import java.awt.*; and simkit.data.*; You will also have to download the CloseableDataWindow class from the course web site.
5. You will have to write the constructors as well; they should have the same signatures as TriangularVariate and just pass their arguments to super().

Output

Histograms that should (hopefully) look roughly like the triangular pdf.

Deliverables

Turn in your source code and a picture of your histograms. To print a picture, select the window with the histogram and press <ALT>-Print Screen. Then open up Wordpad (or Word, if you must) and paste the picture into the document. Finally, print the document.